**Publish and Subscribe Events in Spring Boot – 2022**

**Create Simple Event class**

@Data

**public** **class** PersonEvent **extends** ApplicationEvent {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **long** id;

**private** String name;

**public** PersonEvent(Object source) {

**super**(source);

}

}

**You can also create a class without extending ApplicationEvent.**

@Data

**public** **class** PersonEvent {

**private** String name;

}

**Create Event Listener**

@Service

**public** **class** EventConsumerService {

**@EventListener**

**public** **void** consumeEvent(PersonEvent data) {

System.***out***.println("Received Event for processing: "+data);

}

**@EventListener**

**public** **void** processPerson(PersonEvent pe) {

System.***out***.println("Person for processing: "+pe);

}

}

**Create Event Publisher**

@Service

**public** **class** EventPublisherService {

**@Autowired**

**private ApplicationEventPublisher publisher;**

**public** **void** performOperation() {

PersonEvent dataEvent = **new** PersonEvent (**this**);

dataEvent.setId(123);

dataEvent.setName("John Abraham");

publisher.**publishEvent**(dataEvent);

}

**public** **void** createPerson() {

PersonEvent pe = **new** PersonEvent();

pe.setName("Ramana Maharshi");

publisher.publishEvent(pe);

}

}

**A simple autostart to test the application**

@Component

**public** **class** AutoStart {

@Autowired

**private** EventPublisherService pubService;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** afterStartup() {

pubService.performOperation();

pubService.createPerson();

}

}

**Spring Boot Main Application**

@SpringBootApplication

**public** **class** EventPubSubMain {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EventPubSubMain.**class**, args);

}

}

**Maven pom.xml**

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

<relativePath />

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

Better way of Defining Event in Spring Boot-2024

If the order is cancelled, event should be published and action will be performed to cancel and to close the event.

@Data  
**public class** Order {  
  
 **private** String name;  
 **private** String status;  
  
 **public** Order(String name) {  
 **this**.name = name;  
 }  
}

@Data  
**public class** CancelledOrderEvent {  
  
 **private** Order order;  
  
 **public** CancelledOrderEvent(Order order) {  
 **this**.order = order;  
 }  
}

@Component  
**public class** CancelledEventListner {  
  
 @EventListener  
 **public void** cancellAndClose(CancelledOrderEvent event) {  
 Order order = event.getOrder();  
 System.***out***.println("Order is cancelled and closed for operation ..."+order.getName());  
 }  
}

**public interface** OrderService {  
  
 **void** cancellOrder(Order order);  
}

@Service  
**public class** OrderServiceImpl **implements** OrderService {  
  
 @Autowired  
 **private** ApplicationEventPublisher eventPublisher;  
  
 @Override  
 **public void** cancellOrder(Order order) {  
 **if**(order.getStatus().equalsIgnoreCase("cancelled")) {  
 CancelledOrderEvent event = **new** CancelledOrderEvent(order);  
 eventPublisher.publishEvent(event);  
 }  
 }  
}

@Component  
**public class** AutoRun {  
  
 @Autowired  
 **private** OrderService orderService;  
  
 @EventListener(ApplicationReadyEvent.**class**)  
 **public void** run() {  
 System.***out***.println("Application running ...");  
 // Get the order details from Database  
 Order order = **new** Order("Samsung Mobile");  
 order.setStatus("cancelled");  
 orderService.cancellOrder(order);  
 }  
}

**Usage of @Order Annotation in Spring**

@Component  
public class ApplicationStartUp {  
  
 @Autowired  
 private List<Validation> validationList;  
  
 **@EventListener(ApplicationReadyEvent.class)**  
 public void afterStartup() {  
 System.*out*.println("Application Started at : " + CommonUtil.*timeNow*());  
 check();  
 }  
  
 public void check() {  
 for(int i = 0; i < validationList.size(); i++) {  
 System.*out*.println(validationList.get(i).validate());  
 }  
 }  
}

public interface Validation {  
 String validate();  
}

**Validations in Order**

@Component  
@Order(1)  
public class PanCardValidation implements Validation {  
 @Override  
 public String validate() {  
 return "Success";  
 }  
}

@Component  
@Order(2)  
public class AadharValidationImpl implements Validation {  
 @Override  
 public String validate() {  
 return "Sucess";  
 }  
}

@Component  
@Order(3)  
public class VoterIdValidationImpl implements Validation {  
 @Override  
 public String validate() {  
 return "Sucess";  
 }  
}